Introdução ao Escalonamento e Aplicações

Escalonamento com reservas

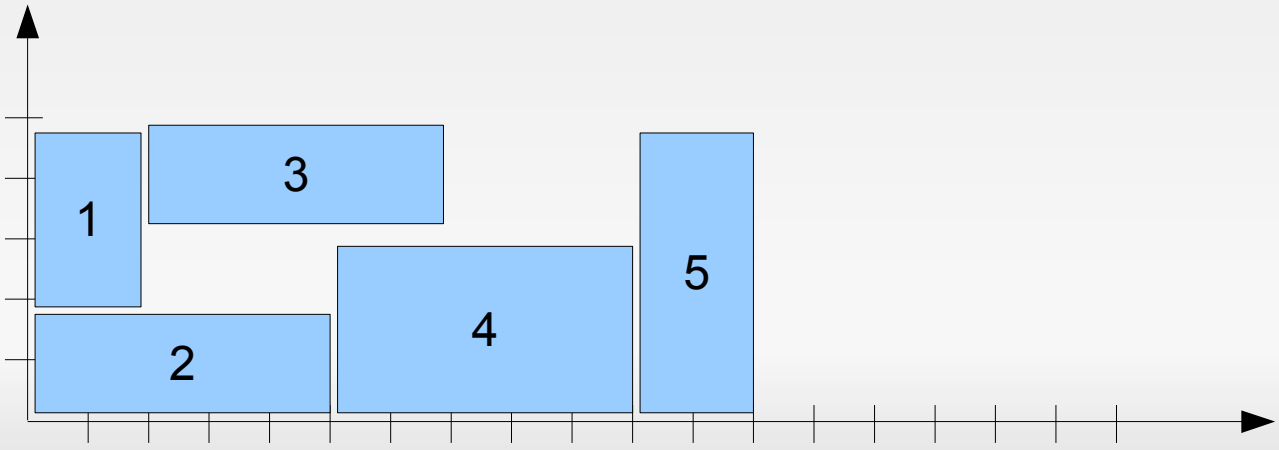
Vinicius Gama Pinheiro

Problema

- Escalonamento de um conjunto de processos independentes sobre um conjunto de processadores homogêneos com restrições de disponibilidade (reservas)
  - Essas reservas representam blocos de tempo e quantidade de recursos agendados para outras atividades
  - Nesses períodos de tempo os recursos ficam indisponíveis para o processamento de tarefas

Preliminares

- RIGIDSCHEDULING (modelo clássico)
  - Escalonar *n* aplicações independentes em *m* processadores idênticos
  - Cada aplicação *j* requer um número fixo q_j de processadores ($q_j \in [1..m]$, para $1 \leq j \leq n$)
  - $P \mid p_j, \text{tam}_j \mid C_{max}$ (tempo de execução $p_j > 0$)

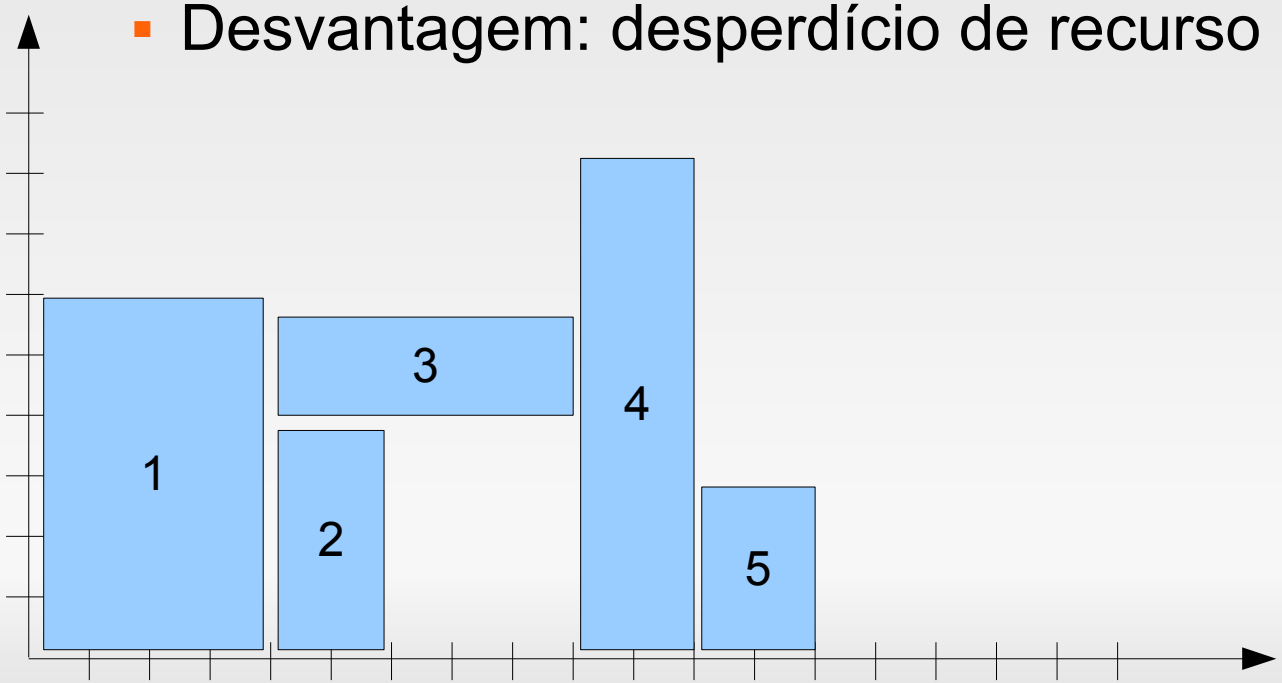


Preliminares

- Solução: n tempos de início (σ_i), $i = 1..n$
  - Para todo $t \geq 0$, $\sum q_i \leq m$, com i $\in I_t$
  - $I_t = \{ i \in [1..n] \mid \sigma_i \leq t < \sigma_i + p_i \}$
  - Minimizar $C_{max} = \max_{i \in [1..n]}(\sigma_i + p_i)$
- Problema é NP-difícil
- O problema de escalonar tarefas sequenciais em dois processadores já é fracamente NP-difícil (problema da partição)
- O escalonamento rígido é fortemente NP-difícil mesmo quando m é fixado com um valor maior ou igual a 5

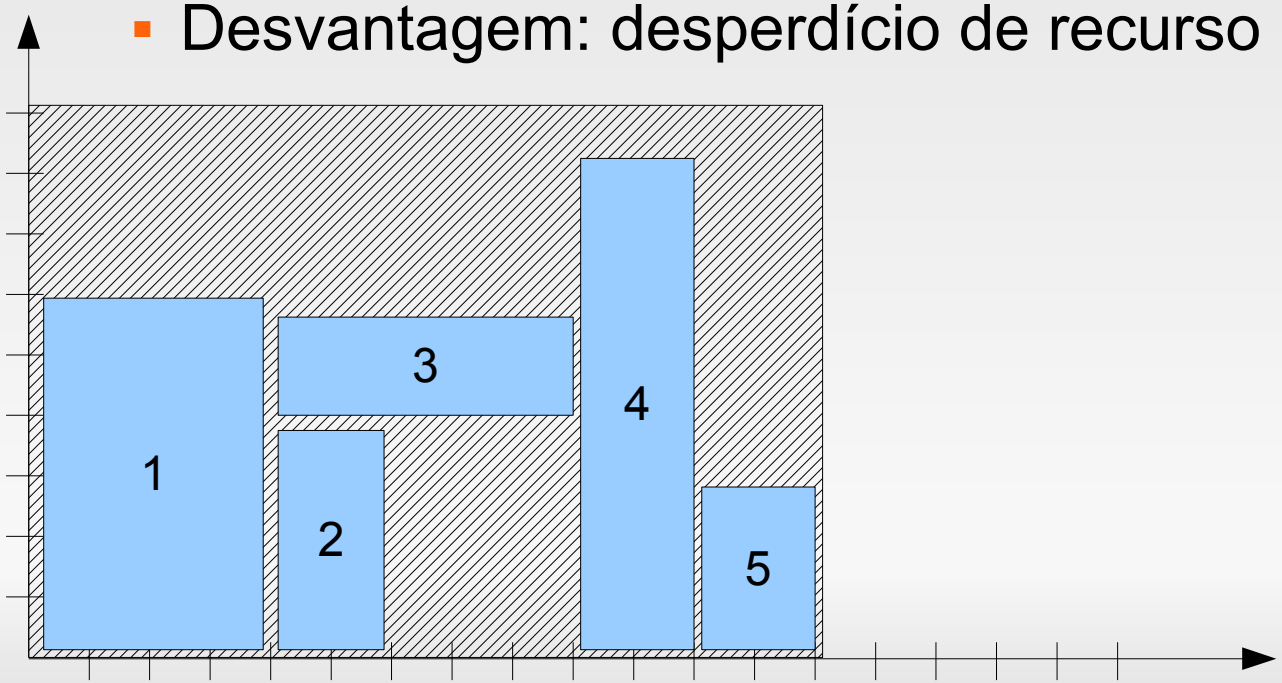
Preliminares

- Algoritmos clássicos
  - First Come First Serve (FCFS)
    - Escalonamento guloso (em batch): escalona a próxima tarefa assim que houver recursos suficientes
    - Desvantagem: desperdício de recurso



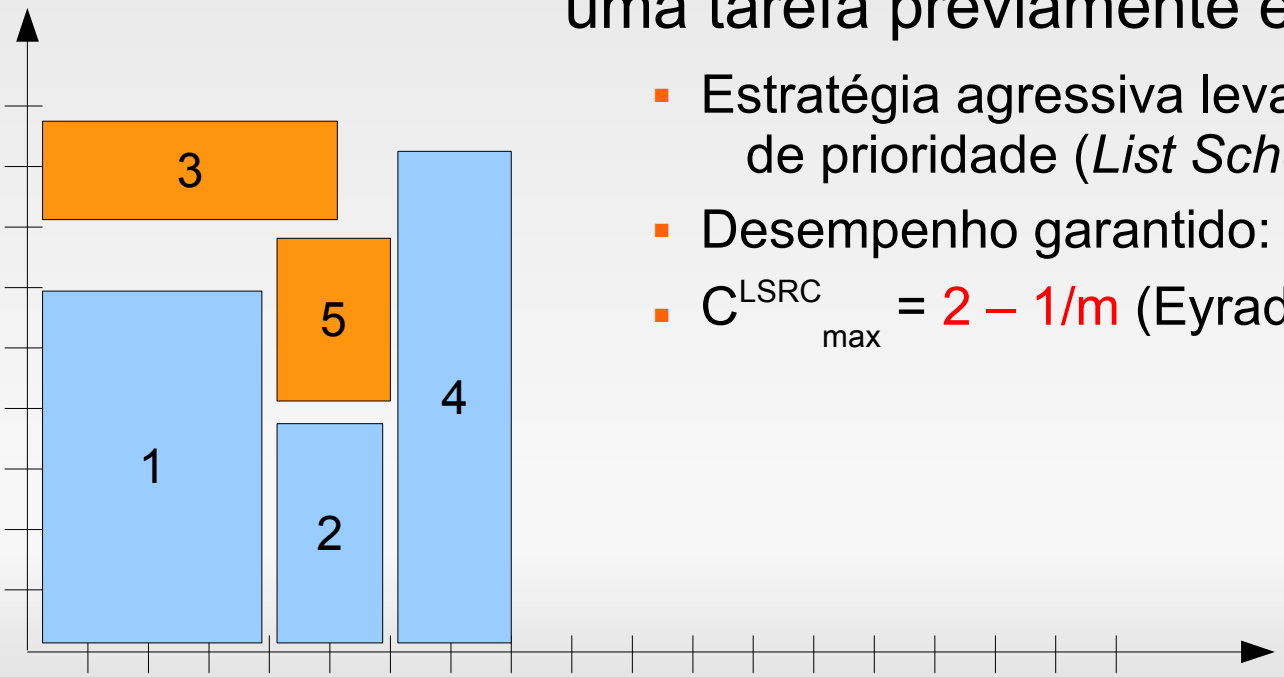
Preliminares

- Algoritmos clássicos
  - First Come First Serve (FCFS)
    - Escalonamento guloso (em batch): escalona a próxima tarefa assim que houver recursos suficientes
    - Desvantagem: desperdício de recurso



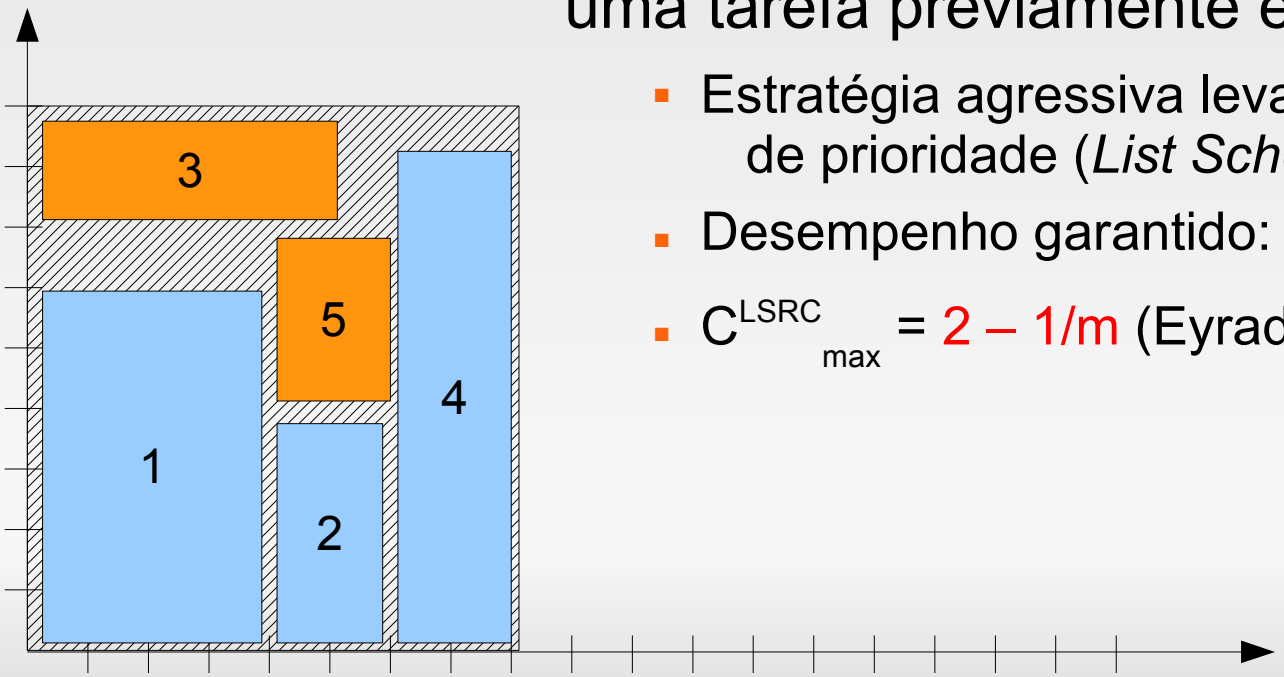
Preliminares

- Algoritmos clássicos
  - Estratégias de FCFS com back-filling
    - Conversadora: não atrasa o início de nenhuma tarefa previamente escalonada
    - Agressiva: qualquer tarefa pode atrasar o início de uma tarefa previamente escalonada
      - Estratégia agressiva leva a um algoritmo com regras de prioridade (*List Scheduling*)
      - Desempenho garantido: 2 (Garey, Graham)
      - $C^{LSRC}_{max} = 2 - 1/m$ (Eyrad-Dubois et al)



Preliminares

- Algoritmos clássicos
  - Estratégias de FCFS com back-filling
    - Conversadora: não atrasa o início de nenhuma tarefa previamente escalonada
    - Agressiva: qualquer tarefa pode atrasar o início de uma tarefa previamente escalonada
      - Estratégia agressiva leva a um algoritmo com regras de prioridade (*List Scheduling*)
      - Desempenho garantido: $C^{LS}_{max} = 2$ (Garey, Graham)
      - $C^{LSRC}_{max} = 2 - 1/m$ (Eyrad-Dubois et al)

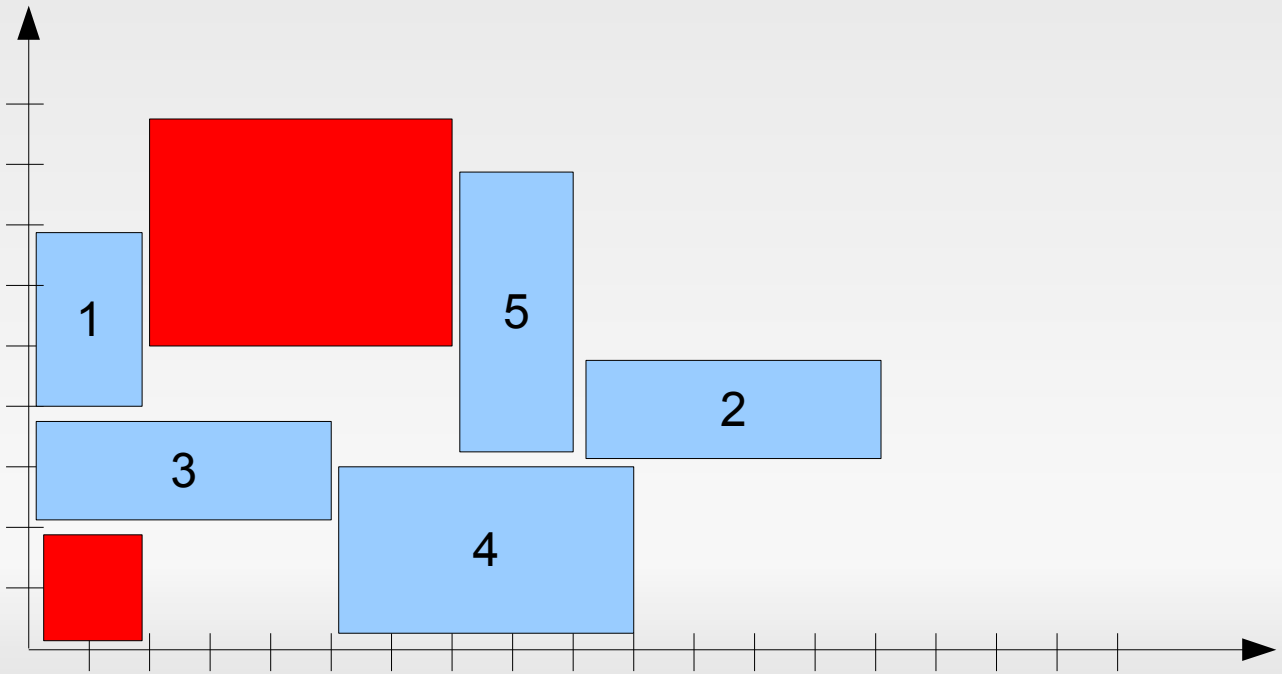


Escalonamento com reservas

- RESASCHEDULING
  - Um inteiro *m* (número de processadores)
  - Um conjunto de *n* de aplicações independentes T_i ($i = 1..n$) cada uma com duração $p_i > 0$ e que requer $q_i \in [1..m]$ processadores
  - *n'* reservas R_j ($j = n+1..n+n'$) cada uma com duração com início em $r_j > 0$, duração $p_j > 0$ e que requer $q_j \in [1..m]$ processadores

Escalonamento com reservas

- RESASCHEDULING
  - Problema: minimizar o C_{max}
  - Ambos os modelos não consideram contiguidade!
  - Offline pode ser usado como online (dobra ρ)



Escalonamento com reservas

- Considerar apenas instâncias viáveis
  - Para todo $t \geq 0$, $\sum q_j \leq m$, com $j \in J_t$
  - $J_t = \{ j \in [n+1..n+n'] \mid r_j \leq t < r_j + p_j \}$
  - A partir disso podemos gerar a função de indisponibilidade como $U(t) = \sum q_j \leq m$, com $j \in J_t$ que representa o número de máquinas indisponíveis no tempo t
  - Sendo assim, para todo t, $U(t) \leq m$
- Solução: n tempos de início (σ_i), $i = 1..n$
  - Para todo $t \geq 0$, $\sum q_i \leq m - U(t)$, com i $\in I_t$

Escalonamento com reservas

- Análise
  - NP-difícil: contém o modelo clássico (n' = 0)
  - Impossível definir um Algoritmo de aproximação em tempo polinomial
    - Ex.: é possível inserir uma reserva muito larga e longa que começa no valor ótimo de C_{max}
    - Não perturba a solução ótima, mas leva a C_{max} arbitrários para qualquer escalonamento não ótimo

Escalonamento com reservas

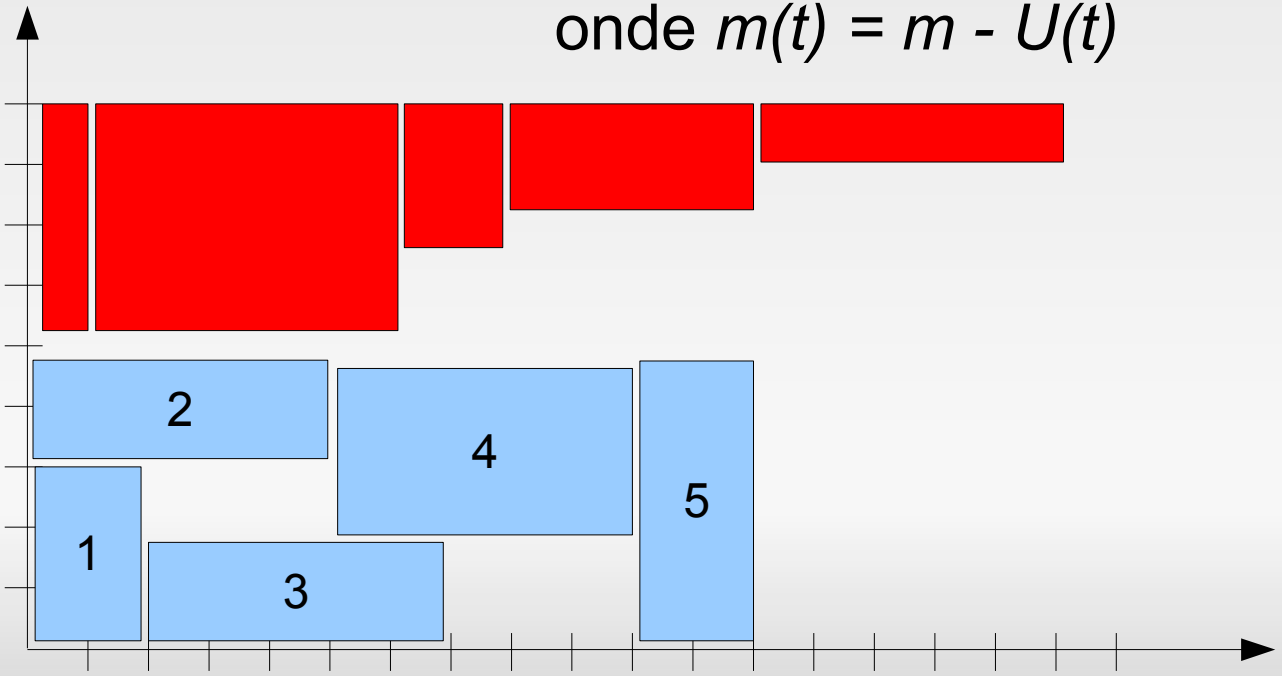
- Teorema 1
  - Se P ≠ NP, não há algoritmo polinomial para o RESASCHEDULING com uma taxa de desempenho finita, mesmo no caso restrito de m=1 ou n'=1
- Prova
  - n' = 1, redução a partir de RIGIDSCHEDULING
  - Para m = 1, redução a partir de 3-partição
  - $C^*_{max} = k(B + 1) - 1$ (instância com 1 máquina na qual B é o tempo entre duas reservas e 3k é o número de tarefas)

Escalonamento com reservas

- Reservas com restrições
  - Reservas não-crescentes
    - Proposição 1: para cada instância I com reservas não-crescentes, temos que

$$C^{LSRC}_{max} \leq (2 - 1/m(C^{*}_{max}))C^{*}_{max} \leq (2 - 1/m)C^{*}_{max},$$

onde $m(t) = m - U(t)$



Escalonamento com reservas

- Reservas com restrições
  - α-resascheduling: limite de reservas e de máquinas solicitadas
    - Dado $\alpha \in]0;1]$, todas as reservas em um dado tempo t não pode exceder $(1 - \alpha)m$ máquinas e tarefas não solicitam mais do que αm máquinas
    - Para todo $t \geq 0$, $U(t) = \sum q_j \leq (1 - \alpha)m$, com $j \in J_t$
    - Para todo $i \leq n$, $q_i \leq \alpha m$
  - Proposição 2 (lower bound): Se $2/\alpha$ é um inteiro, a garantia de desempenho do LSRC é pelo menos $\underline{2/\alpha - 1 + \alpha/2}$
  - Proposição 3 (upper bound): para o problema do α-resascheduling LSRC tem uma garantia $p \leq \underline{2/\alpha}$

Escalonamento com reservas

- Reservas com restrições
  - **Proposição 2 (lower bound): Se $2/\alpha$ é um inteiro, a garantia de desempenho do LSRC é pelo menos**

$$2/\alpha - 1 + \alpha/2$$

    - **Prova:** constrói uma instância para o caso $\alpha = 2/k$, onde k é um inteiro no qual o escalonamento ótimo usa m máquinas praticamente todo o tempo, mas para o LSRC usar apenas αm máquinas

Escalonamento com reservas

- Reservas com restrições
  - **Proposição 3 (upper bound): para o problema do α-resascheduling LSRC tem uma garantia**

$$p \leq 2/\alpha$$

  - **Prova:** a prova revisita o limite estabelecido por Graham, definindo processadores como o único recurso compartilhado

Obrigado!

Perguntas?